47장 에러 처리

47.1 에러 처리의 필요성

- 발생한 에러에 대해 대처하지 않고 방치하면 프로그램은 강제 종료된다.
  - `try...catch` 문을 사용해 발생한 에러에 적절하게 대응하면 프로그램이 강제 종료되지 않고 계속해서 코드를 실행시킬 수 있다.

```
console.log('[Start]');

try {
    foo();
} catch (error) {
    console.error('[에러 발생]', error);
    // [에러 발생] ReferenceError: foo is not defined
}

// 발생한 에러에 적절한 대응을 하면 프로그램이 강제 종료되지 않는다.
console.log('[End]');
```

- 작성한 코드에서는 언제나 예외적인 상황이 발생할 수 있다는 것을 전제하고

  이에 대응하는 코드를 작성하는 것이 중요하다.

47.2 try...catch...finally 문

- 에러 처리를 구현하는 두 가지 방법

1. 예외적인상황이 발생하면 반환하는 값을 if 문이나 단축 평가 또는 옵셔널 체이닝 연산자를 통해 확인해서 처리하는 방법
2. 에러 처리 코드를 미리 등록해 두고 에러가 발생하면 에러 처리 코드로 점프하도록 하는 방법
   - `try...catch...finally`
   - 일반적으로 이 방법을 에러 처리라고 한다.

- `try...catch...finally` 문은 3개의 코드 블록으로 구성된다.

```
try {
    // 실행할 코드(에러가 발생할 가능성이 있는 코드)
} catch (err) {
    // try 코드 블록에서 에러가 발생하면 이 코드 블록의 코드가 실행된다.
  // err에는 try 코드 블록에서 발생한 Error 객체가 전달된다.
} finally {
    // 에러 발생과 상관없이 반드시 한 번 실행된다.
}
```

47.3 Error 객체

- Error 생성자 함수는 에러 객체를 생성한다.
- Error 생성자 함수에는 에러를 상세히 설명하는 에러 메시지를 인수로 전달할 수 있다.

```
const error = new Error('invalid');
```

- Error 생성자 함수가 생성한 에러 객체는 **message 프로퍼티**와 **stack 프로퍼티**를 갖는다.
- message 프로퍼티의 값은 Error 생성자 함수에 인수로 전달한 에러 메시지이고, stack 프로퍼티의 값은 에러를 발생시킨 콜스택의 호출 정보를 나타내는 문자열이며 디버깅 목적으로 사용한다.
    - Error 생성자 함수를 포함해 7가지의 에러 객체를 생성할 수 있는 Error 생성자 함수를 제공

생성자 함수	인스턴스
Error	일반적 에러 객체
SyntaxError	자바스크립트 문법에 맞지 않는 문을 해석할 때 발생하는 에러 객체
ReferenceError	참조할 수 없는 식별자를 참조했을 때 발생하는 에러 객체
TypeError	피연산자 또는 인수의 데이터 타입이 유효하지 않을 때 발생하는 에러 객체
RangeError	숫자값의 허용 범위를 벗어났을 때 발생하는 에러 객체
URIError	encodeURI 또는 decodeURI 함수에 부적절한 인수를 전달했을 때 발생하는 에러 객체
EvalError	eval 함수에서 발생하는 에러 객체

【 예제 47-08 】

```
1 @ 1;    // SyntaxError: Invalid or unexpected token
foo();    // ReferenceError: foo is not defined
null.foo; // TypeError: Cannot read property 'foo' of null
new Array(-1); // RangeError: Invalid array length
decodeURIComponent('%'); // URIError: URI malformed
```

47.4 throw 문

- Error 생성자 함수로 에러 객체를 생성한다고 에러가 발생하는 것은 아니다.
  - 즉, 에러 객체 생성과 에러 발생은 의미가 다르다.
- 에러를 발생시키려면 try 코드 블록에서 throw 문으로 **에러 객체를 던져야 한다.**

> 💡 **throw** 표현식;

```
try {
    // 에러 객체를 던지면 catch 코드 블록이 실행되기 시작한다.
    throw new Error('something wrong');
} catch (error) {
    console.log(error);
}
```

47.5 에러의 전파

- 에러는 호출자 방향으로 전파된다.

- 즉, 콜 스택의 아래 방향(실행 중인 실행 컨텍스트가 푸시되기 직전에 푸시된 실행 컨텍스트 방향)으로 전파된다.

```
const foo = () => {
    throw Error('foo에서 발생한 에러'); // (4)
};

const bar = () => {
    foo(); // (3)
};

const baz = () => {
    bar(); // (2)
};

try {
    baz(); // (1)
} catch (err) {
 console.log(err);
}
```

그림 47-1 에러는 호출자 방향으로 전파된다.

💡 비동기 함수인 setTimeout이나 프로미스 후속 처리 메서드의 콜백 함수는 호출자가 없다.

setTimeout이나 프로미스 후속 처리 메서드의 콜백 함수는 태스크 큐나 마이크로태스크 큐에 일시 저장되었다가 콜 스택이 비변 이벤트 루프에 의해 콜 스택으로 푸시되어 실행되는데, 이때 콜 스택에 푸시된 함수의 실행 컨텍스트는 콜 스택의 가장 하부에 존재하게 된다.

따라서 에러를 전파할 호출자가 존재하지 않게 된다.